Approximation polyédrique : l’exemple d’OpenGl

Frank Singhoff

Bureau C-203

Université de Brest, France

LISyC/EA 3883

singhoff@univ-brest.fr

Sommaire

Introduction (1)

- **OpenGl, c'est :**
  - C'est une bibliothèque graphique 2D/3D avec une API "à état"'.
  - Conçue en 1989 (GL) par Sillicon Graphics, puis portée sur d'autres architectures en 1993 (OpenGL).
  - Avec OpenGL, on peut produire des images synthétiques sophistiquées (reflets, ombres) en temps réel si on dispose de ressources matérielles adéquates.
  - Existe sur de nombreuses architectures (NT, Unix, OS2, Amiga, etc...)
  - Nombreuses implémentations (Exemple sous NT : SGI OpenGL, 3dfx OpenGL, Mesa, Microsoft OpenGL, ...)

Introduction (2)

- Quelles bibliothèques avec OpenGL ?

1. La bibliothèque OpenGL à proprement dit.
2. Des bibliothèques complémentaires :
   - **OpenGL Utility Library (GLU) :** surcouche à OpenGL, fournit des fonctions plus évoluées pour la gestion des caméras ou la création de modèle avec une haute définition.
   - **OpenGL Extension Library pour le système graphique X11 (GLX) :** interopérabilité avec le protocole X11.
   - **GL Utility Toolkit (GLUT)** : fournit une interface simple pour le dialogue avec le système d'exploitation.

Introduction (3)

Suffixe	OpenGl	C
b	GLbyte	signed char
i	GLint	int
f	GLfloat	float
d	GLdouble	double

- Signatures de l'API :

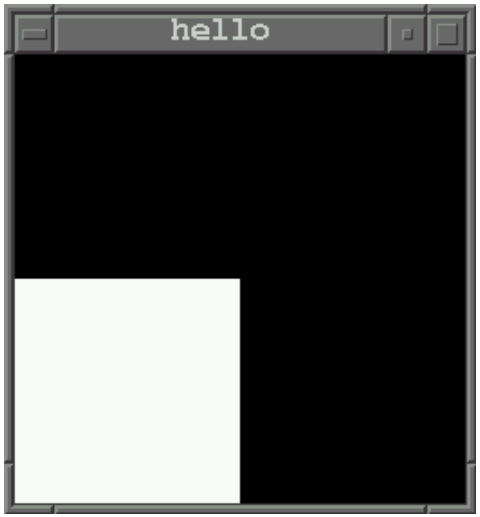
`[xxx]service[n][y](a0, a1, a2, ..., an);` avec xxx pour la bibliothèque et y pour le suffixe.

- Exemples :

```
glVertex2i(1, 3); // sommet (1,3,0)
glVertex3f(1.0, 3.0, 0.0);
```

Introduction (4)

- Dessiner notre première scène :



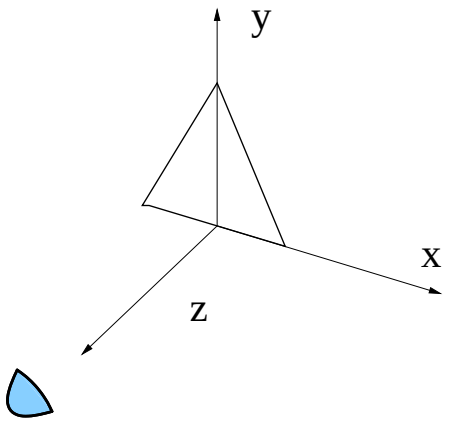
- Dessiner cette scène, c'est :

1. Vider l'écran, c-a-d les tampons.
2. Choisir une couleur (le blanc).
3. Dessiner le carré (c'est un polygone).

Sommaire

Approximation polyédrique (1)

- Comment mémoriser un volume ? un objet 3D ?

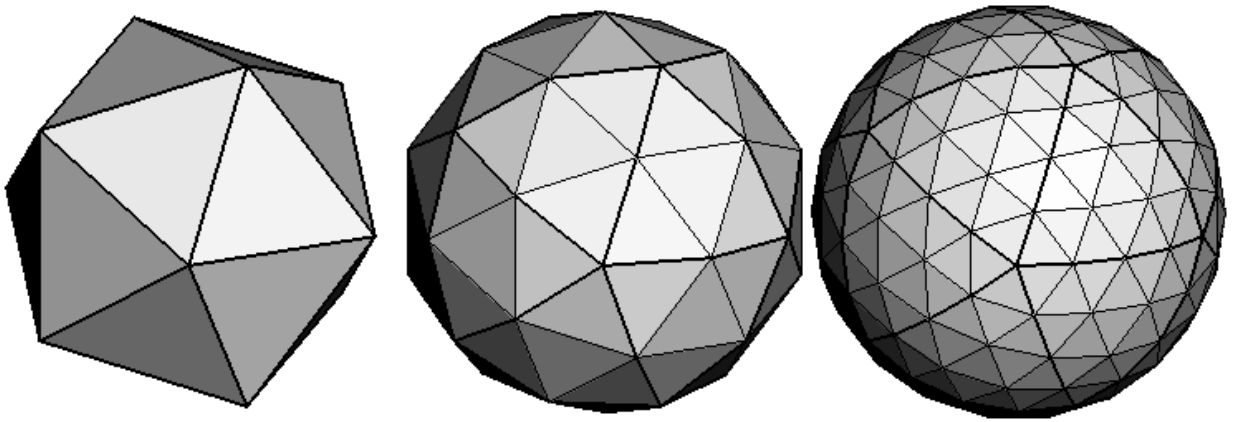


Coordonnées = (x,y,z)

- Chaque sommet est défini par un triplé (x,y,z) le situant dans un repère cartésien.
- Un objet 3D est approximé par une série (finie) de sommets calculés ou non. Les sommets peuvent constituer soit :
  - Des polyèdres (ensemble de sommets constituant des faces, ou polygones).
  - Des courbes ou surfaces d'interpolation (ex : Beziers ou B-splines).

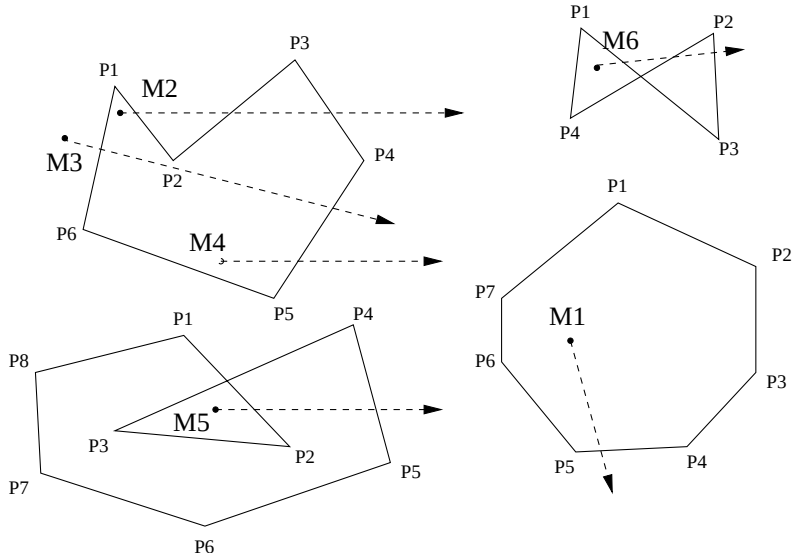
Approximation polyédrique (2)

- Exemple : la sphère



- Nécessaire compromis entre efficacité et qualité de l'image (méthode d'approximation) :
  - Nombre de sommets et stockage en mémoire.
  - Temps de calcul pour les animations (ex : translation) ou les effets graphiques (ex : lumière).

Approximation polyédrique (3)



- **Polygone dans un plan :** est défini par une suite de segments $s1, ..., sn$ tel que le segment $si = [Pi, Pi + 1]$ où Pi est un point du plan avec P1=Pn. $i = 1, ..., n$ s'appellent les "sommets" du polygone. Les segments si sont nommés "arêtes".
- **Point intérieur :** soit un polygone P est un point M du plan. M est intérieur si la demie-droite issue de M intersecte un nombre impair de segments de P.

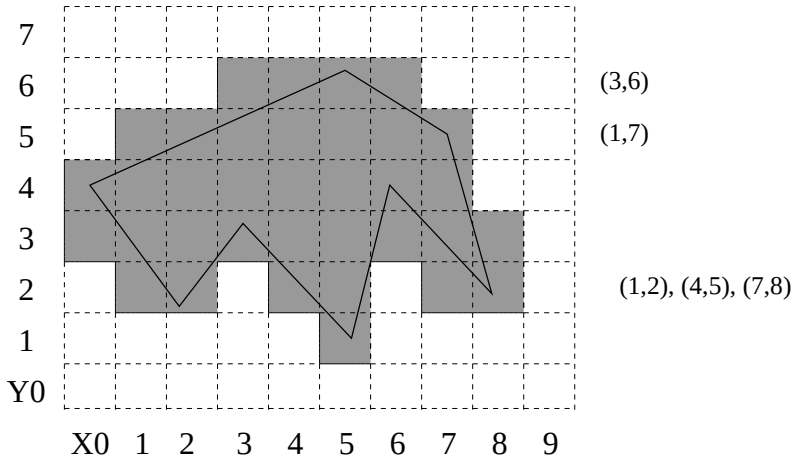
Approximation polyédrique (4)

- **Polygone convexe :** un polygone P est convexe si
  1. Pour tous points M et M' qui sont intérieurs à P, le segment [M,M'] est entièrement composé de points intérieurs à P.
  2. Toutes ses diagonales (segment qui joint 2 sommets non consécutifs) sont entièrement à l'intérieur de la surface délimitée par le polygone.

- **Polygone concave :** P est concave si l'une de ses diagonales n'est pas entièrement à l'intérieur de la surface délimitée.

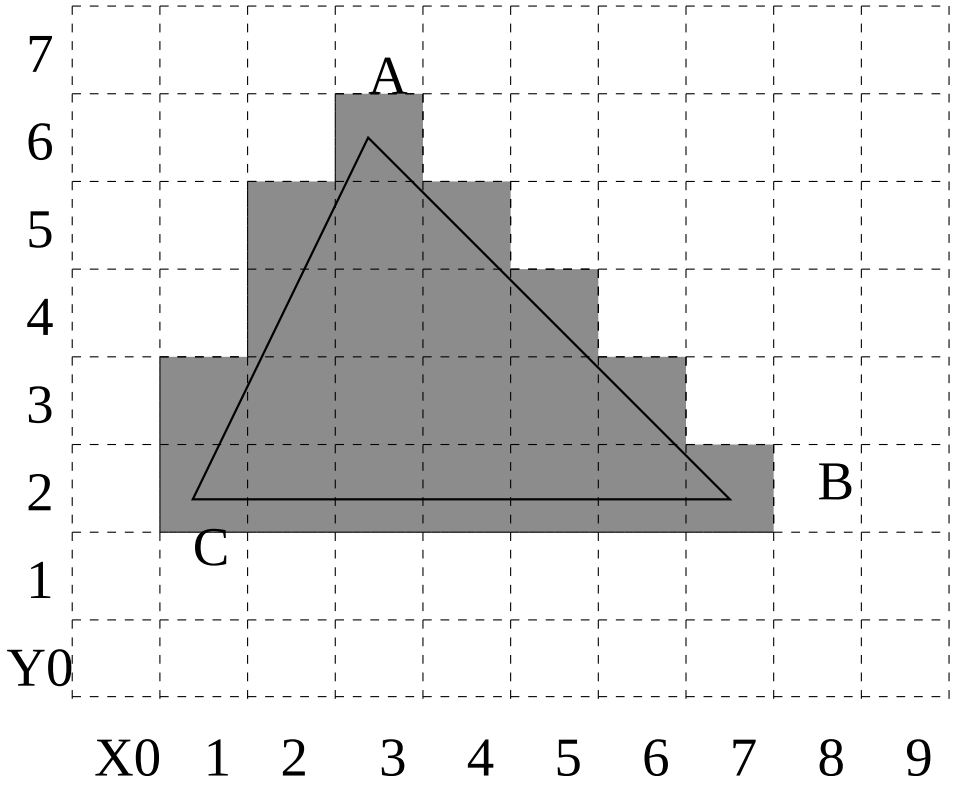
- **Polygone avec autointersection:** P contient une autointersection si deux de ses segments non consécutifs s'intersectent.

Approximation polyédrique (5)



- Algorithme de remplissage naïf :
  1. Pour chaque pixel, tester par parité son appartenance au polygone.
  2. Colorier le pixel selon (1).
- Exemple d'algorithme de remplissage de polygones de type "scanline" :
  1. Traiter le polygone ligne par ligne.
  2. Parcourir les arêtes du polygone et détecter les segments intérieurs.
  3. Colorier segments par segments.

Approximation polyédrique (6)

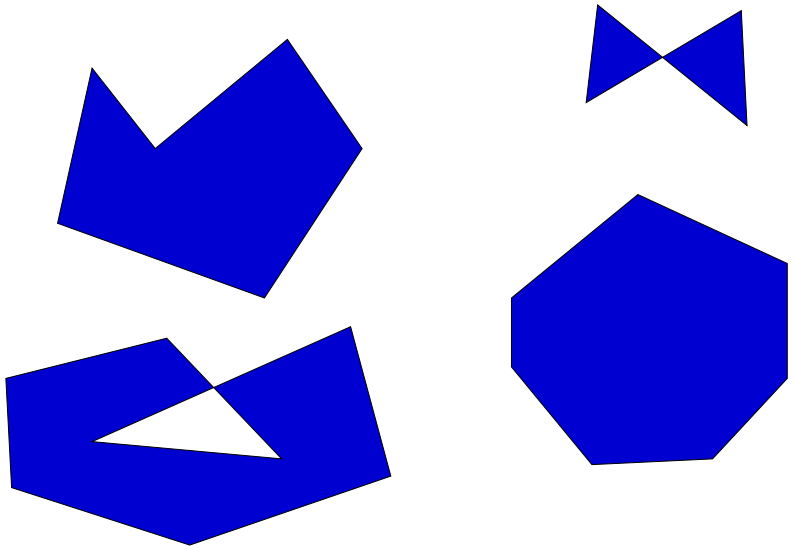


Xmin Xmax

AB	BC	CA
(+inf, −inf)	(+inf, −inf)	(+inf, −inf)
(+inf,3)	(+inf,3)	(3,3)
(+inf,4)	(+inf,4)	(2,4)
(+inf,5)	(+inf,5)	(2,5)
(+inf,6)	(+inf,6)	(1,6)
(+inf,7)	(1,7)	(1,7)
(+inf, −inf)	(+inf, −inf)	(+inf, −inf)
(+inf, −inf)	(+inf, −inf)	(+inf, −inf)

Approximation polyédrique (7)

- Polygones remplis :



- Attention : beaucoup de bibliothèques (c'est le cas pour OpenGL) supposent que les polygones sont convexes !!!
- Des polygones concaves, à trous, avec autointersection peuvent être redécoupés en sous-polygones.

Approximation polyédrique (8)

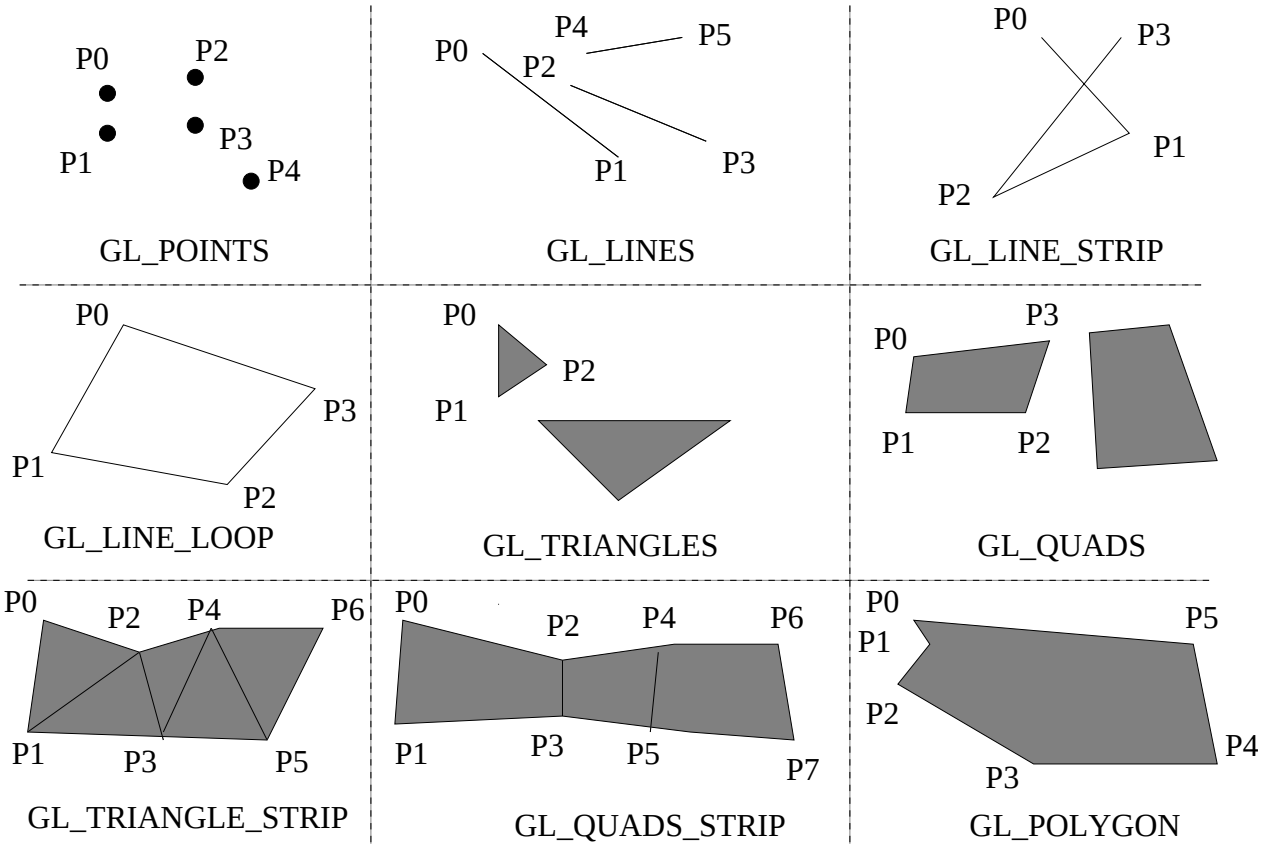
- Comment définit-on un polygone avec OpenGl :

1. Par une liste **ordonnée** de sommets. Un sommet est défini par la fonction $glVertex*$. Une liste de sommets est définie par les fonctions $glBegin$ et $glEnd$. **Open GL ne supporte que les polygones convexes**.
2. Par une méthode de connection de ces sommets (ou primitives).

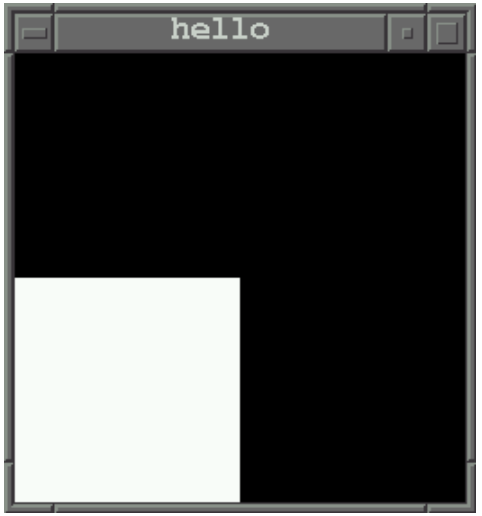
- Primitives disponibles : GL_POINTS, GL_LINES, GL_LINE_STRIP, $GL\_TRIANGLES$, $GL\_TRIANGLES\_STRIP$, $GL\_TRIANGLE\_FAN$, GL_QUADS, GL_QUADS_STRIP, $GL\_POLYGON$.

Approximation polyédrique (9)

- Primitives OpenGL pour tracer des points/lignes/polygones :



Approximation polyédrique (10)



- Dessiner cette scène, c'est :

1. Vider l'écran, c-a-d les tampons.
2. Choisir une couleur (le blanc).
3. Dessiner le carré (c'est un polygone).

Approximation polyédrique (11)

- Exemple de fonction d'affichage d'une scène :

```
/* Dessine le polygone */
   glBegin(GL_POLYGON);
      glVertex3f (0.0, 0.0, 0.0);
      glVertex3f (0.5, 0.0, 0.0);
      glVertex3f (0.5, 0.5, 0.0);
      glVertex3f (0.0, 0.5, 0.0);
   glEnd();
```

- Quelle est la dimension du polygone ?
- Quelle est la couleur du polygone ?

Sommaire

Gestion de la couleur (1)

- Deux principaux codes existent : le RVB (ou RGB, RGBA, RVBA) et luminance/chrominance :

1. **Luminance/chrominance.** Développé à la base pour la télévision hertzienne (passage du noir/blanc à la couleur). Code YIQ (NTSC) et code YUV (PAL/SECAM). Intensité de lumiere (Y) ; Information chromatique : IQ/UV.
2. **RVBA.** Chaque couleur est codée par 4 composantes qui définissent l'intensité en R(rouge), V(vert), B(bleue), A(alpha).

- Conversion possible entre RVB/YIQ/YUV.

Gestion de la couleur (2)

- Dans OpenGl, deux modes de mémorisation de la couleur des pixels existent :

1. **Mode RVBA direct :** le buffer mémorise directement pour chaque pixel la couleur sous la forme des composantes RVBA.
2. **Mode RVBA index :** le buffer mémorise pour chaque pixel un index dans une table de couleur RVBA.

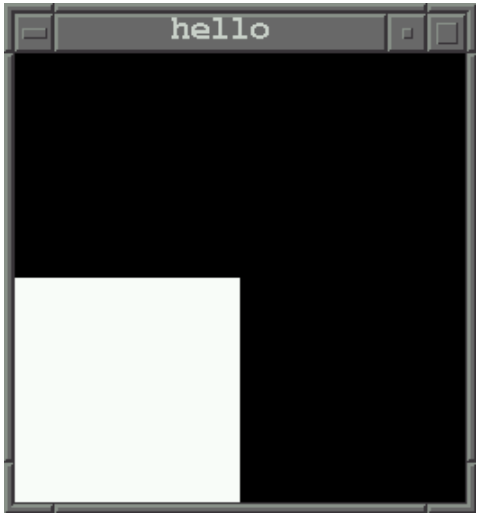
- Quel mode choisir ? bitplans disponibles, nombre de couleurs nécessaires, taille tampons chromatiques, services OpenGL à utiliser.

Gestion de la couleur (3)

- $glColor$ modifie la variable d'état définissant la couleur active dans le mode RVBA.
- Les valeurs de chaque composante sont comprises entre 0 et 1.
- Quelques exemples de valeurs significatives :

```
glColor3f(0.0, 0.0, 0.0);   // noir
glColor3f(0.0, 0.0, 1.0);   // bleu
glColor3f(0.0, 1.0, 0.0);   // vert
glColor3f(1.0, 0.0, 0.0);   // rouge
glColor3f(0.7, 0.7, 0.7);   // gris clair
glColor3f(0.3, 0.3, 0.3);   // gris fonce
glColor3f(1.0, 1.0, 1.0);   // blanc
glColor3f(1.0, 1.0, 0.0);   // jaune
glColor3f(1.0, 0.6, 0.6);   // rose
```

Gestion de la couleur (4)



- Dessiner cette scène, c'est :

1. Vider l'écran, c-a-d les tampons.
2. Choisir une couleur (le blanc).
3. Dessiner le carré (c'est un polygone).

Gestion de la couleur (5)

- Exemple de fonction d'affichage de notre scène :

```
/* Dessine le polygone en BLANC ! */
   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.0, 0.0, 0.0);
      glVertex3f (0.5, 0.0, 0.0);
      glVertex3f (0.5, 0.5, 0.0);
      glVertex3f (0.0, 0.5, 0.0);
   glEnd();
```

Sommaire

Gestion des tampons (1)

- **Tampons avant (GL_FRONT) et arrière (GL_BACK) suivants :**
  - chromatiques ($GL\_COLOR\_BUFFER\_BIT$).
  - de profondeur ($GL\_DEPTH\_BUFFER\_BIT$).
  - d'accumulation ($GL\_ACCUM\_BUFFER\_BIT$).
  - stencil ($GL\_STENCIL\_BUFFER\_BIT$).

- **Principaux services :**
  1. Vider un tampon : $glClear$.
  2. Sélectionner la valeur pour vider un tampon : $glClearColor$, $glClearDepth$, $glclearStencil$, $glClearAccum$.
  3. Copier/lire/écrire : $glDrawBuffer$, $glReadBuffer$, $glCopyPixels$, ...
  4. Tests tels que lissage, stencil, profondeur, ...

Gestion des tampons (2)

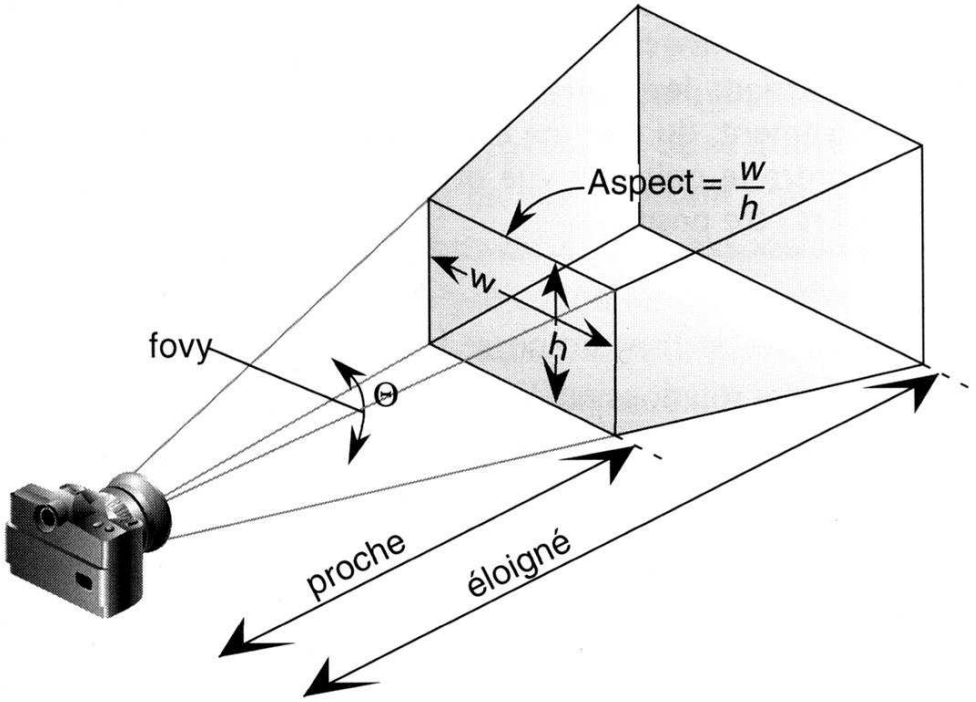
- **Exemple du test de profondeur :**
  - Le test de profondeur permet de **rejeter**/ne pas afficher les polygones cachés par d'autres (opération de clipping).
  - Il n'est pas activé par défaut. Activation par :

$$glEnable(GL\_DEPTH\_TEST)$$

  - Lors de la restitution, le tampon $GL\_DEPTH\_BUFFER\_BIT$ est rempli conformément à la position des différents objets. Entre deux restitutions, ce tampon doit donc être vidé par :

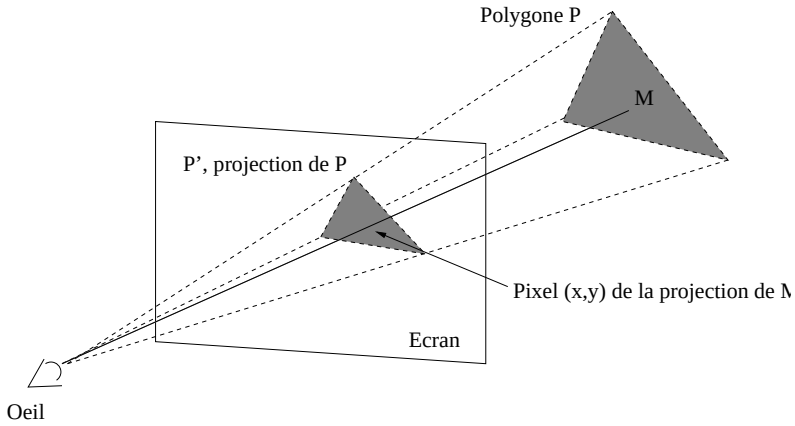
$$glClear(GL\_DEPTH\_BUFFER\_BIT)$$

Gestion des tampons (3)



- **Principe de l'algorithme du Z-buffer :** calculer pour chaque pixel le polygone de profondeur minimum et colorier le pixel avec la couleur du polygone. Traitement itératif (tous les polygones sont analysés).

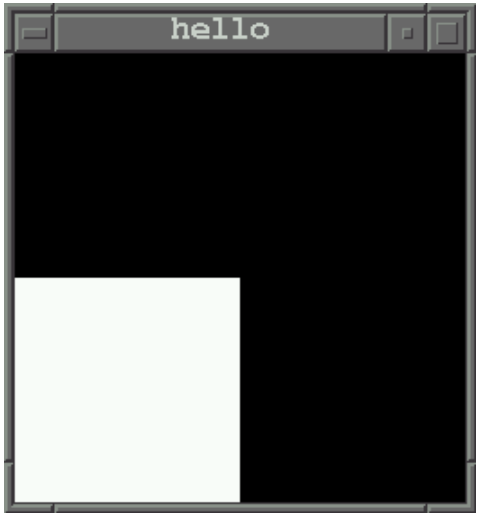
Gestion des tampons (4)




```
int x,y;
double minz[MAX_X][MAX_Y]; // Profondeur courante, initialisée à +inf
double image[MAX_X][MAX_Y]; // Couleur des pixels de l'image

Pour chaque polygone P de la scène
  P'= polygone après projection et clipping de cadrage
  Pour chaque pixel (x,y) de l'image qui soit intérieur à P'
    Soit M, un point de P se projetant sur (x,y);
      if (M.z < minz[x][y])
         image[x][y]=couleur de P en M.
         minz[x][y]=M.z;
```

Gestion des tampons (5)



- Dessiner cette scène, c'est :

1. Vider l'écran, c-a-d les tampons.
2. Choisir une couleur (le blanc).
3. Dessiner le carré (c'est un polygone).

Gestion des tampons (6)

- Exemple de fonction d'affichage de notre scène :

```
/* Vider les tampons chromatique
    et de profondeur */
   glClear(GL_COLOR_BUFFER_BIT);
   glClear(GL_DEPTH_BUFFER_BIT);

/* Dessiner le polygone en BLANC */
   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.0, 0.0, 0.0);
      glVertex3f (0.5, 0.0, 0.0);
      glVertex3f (0.5, 0.5, 0.0);
      glVertex3f (0.0, 0.5, 0.0);
   glEnd();
```

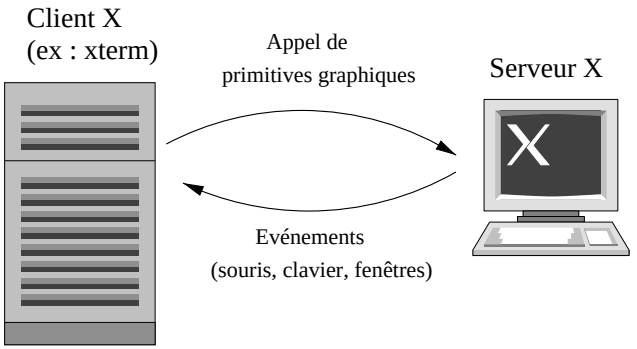
Sommaire

La bibliothèque GLUT

- La GLUT offre des services annexes à l'animation 3D ... mais des services nécessaires :

  - Interaction avec le système de multi-fenêtrage (X11).
  - Interaction avec les périphériques d'entrées (souris/claviers).
  - Boucle d'événements et outils d'animation.
  - Gestion de menus simplifiés.
  - Quelques objets 3D simplifiés.

Interaction avec le serveur X11



- Programmation événementielle : $callbacks$ + boucle d'attente des événements.

- Principales primitives OpenGl associées à X11 :
  - $glutCreateWindow$: création de fenêtre.
  - $glutInitWindowSize$, $glutInitWindowPosition$: positionnement de variables d'état OpenGL.
  - Callbacks de traitements des événements X11 (clavier, souris, événements Map et $Resize$).

Boucle d'événements et d'animation (1)

- **Services nécessaires pour tous types d'animations :**
  - $glutMainLoop$: boucle d'attente des événements.
  - $gluInitDisplayMode()$: définit le mode de fonctionnement de la bibliothèque (simple/double buffering, codage des couleurs, ...).
  - $glutDisplayFunc$: associe un callback aux demandes d'affichage.
  - $glutPostRedisplay()$: provoque une restitution (et donc un ré-affichage).

- **Une animation peut être implantée de diverses manières :**

1. Animation avec tampon simple. $glFlush$ pour forcer la terminaison de la restitution.
2. Animation avec tampon double. $glutSwapBuffers$ pour forcer la terminaison de la restitution et échanger les tampons.

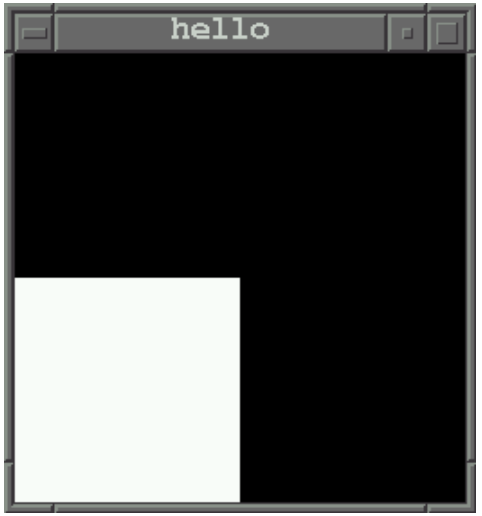
Boucle d'événements et d'animation (2)

- Généralement, les restitutions sont déclenchées sur occurrences d'événements et exécution de leur callback respectif. Pour ce faire, les $callbacks$ invoquent $glutPostRedisplay$ pour provoquer la restitution.

- Exemple de callbacks :

1. Restitution sur clavier/souris : $glutMouseFunc$ ou $glutKeyboardFunc$ enregistrent les callbacks qui vont traiter les événements produits par le périphérique.
2. Restitution sur timer : $glutTimerFunc$ enregistre le callback associé au timer. Le callback est invoqué lorsque le compteur du timer atteint zéro.
3. Restitution par tâche de fond : $glutIdleFunc$.

Boucle d’événements et d’animation (3)



- Dessiner cette scène, c’est :

1. Vider l’écran, c-a-d les tampons.
2. Choisir une couleur (le blanc).
3. Dessiner le carré (c’est un polygone).

Boucle d'événements et d'animation (4)

```
#include <stdlib.h>
#include <GL/glut.h>

void display(void)  {
/* Vider les tampons */
   glClear(GL_COLOR_BUFFER_BIT);
   glClear(GL_DEPTH_BUFFER_BIT);

/* Dessiner le polygone */
   glColor3f (1.0, 1.0, 1.0);
   glBegin(GL_POLYGON);
      glVertex3f (0.0, 0.0, 0.0);
      glVertex3f (0.5, 0.0, 0.0);
      glVertex3f (0.5, 0.5, 0.0);
      glVertex3f (0.0, 0.5, 0.0);
   glEnd();

/* Forcer le rendu maintenant ; glutSwapBuffers
     pour du double buffering*/
   glFlush ();
}
```

Boucle d’événements et d’animation (5)

```
#include <stdlib.h>
#include <GL/glut.h>

/*
  Initialisation de la scène (variables d’état OpenGl)
*/
void init (void)
{
/* Choisir la couleur d’effacement */
   glClearColor (0.0, 0.0, 0.0, 0.0);

/* Selectionner la matrice de projection */
   glMatrixMode(GL_PROJECTION);

/* Initilialise la matrice de projection */
   glLoadIdentity();

/* Clipping de cadrage */
   glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
```

Boucle d’événements et d’animation (6)

- **Restitution déclenchée par le clavier :**

```
/* Callback clavier */
void key(unsigned char k, int x, int y) {
    ...
  glutPostRedisplay();
}

int main(int argc, char** argv) {
   glutInit(&argc, argv);  /* Initialisation de la GLUT */

/* GLUT_SINGLE, GLUT_DOUBLE pour simple/double buffering */
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

/* Création fenêtre et initialisation de la scène */
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello");

/* Connexion callbacks puis boucle d’événements */
   glutDisplayFunc(display); glutKeyboardFunc(key);
   glutMainLoop();
}
```

Boucle d’événements et d’animation (7)

- **Restitution déclenchée par timer :**

```
/* Callback timer */
void my_timer(int x) {
    ...
  glutTimerFunc(40, my_timer, 1);
  glutPostRedisplay();
}

int main(int argc, char** argv) {
/* Initialisation de la GLUT  et choix du mode d’affichage */
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

/* Création fenêtre et initialisation de la scène */
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("hello");

/* Connexion callbacks puis boucle d’événements */
   glutDisplayFunc(display); glutTimerFunc(40, my_timer, 1);
   glutMainLoop();
}
```

Boucle d'événements et d'animation (8)

- Il ne doit pas y avoir d'instruction de restitution hors du callback d'affichage. Appel à $glutPostRedisplay$ pour forcer la restitution.
- Le callback d'affichage doit être idempotent (ne doit pas modifier l'état d'OpenGl).
- Ne pas activer de callback inutilement (performances).
- Attention aux calculs redondants (performances).

Objets 3D de la GLUT

- Sphère pleine ou fil-de-fer :

```
void glutSolidSphere(GLdouble radius,
GLint slices, GLint stacks);
void glutWireSphere(GLdouble radius,
GLint slices, GLint stacks);
```

- Cube plein ou fil-de-fer :

```
void glutSolidCube(GLdouble size);
void glutWireCube(GLdouble size);
```

- Autres objets : Cône, Tore, Tétrahedron, Dodécahedron, Icosahedron, Tasse à thé.

Sommaire

Ce qu’il faut retenir

- Comment décrire un objet par un polyèdre ?
- Comment manipuler couleurs et tampons ?
- Programmation événementielle et animation.